SCALEFUSION: SCALABLE INFERENCE OF SPATIAL-TEMPORAL DIFFUSION TRANSFORMERS FOR HIGH-RESOLUTION LONG VIDEO GENERATION

Jiacheng Yang[*1] **Jun Wu**[2] **Zhen Zhang**[2] **Xinwei Fu**[2] **Zhiying Xu**[2] **Zhen Jia**[2] **Yida Wang**[2]
Gennady Pekhimenko[1]

ABSTRACT

Recent advancements in training diffusion models have made generating high-quality videos possible. Particularly, the spatial-temporal diffusion transformers (ST-DiTs) emerge as a promising diffusion model architecture for generating videos of high-resolution (1080p) and long duration (20 seconds). However, the quadratic scaling of compute cost with respect to resolution and duration, primarily due to spatial-temporal attention layers processing longer sequences, results in high inference latency of ST-DiTs. This hinders their applicability in time-sensitive scenarios. Existing sequence parallelism techniques, such as DeepSpeed-Ulysses and RingAttention, are not optimally scalable for ST-DiT inference across multiple GPU machines due to cross-machine communication overheads. To address this challenge, we introduce ScaleFusion, a scalable inference engine designed to optimize ST-DiT inference for high-resolution, long video generation. By leveraging the inherent structure of spatial-temporal attention layers, ScaleFusion effectively hides cross-machine communication overhead through novel intra-layer and inter-layer communication scheduling algorithms. This enables strong scaling of 3.60× on 4 Amazon EC2 `p4d.24xlarge` machines (32 A100 GPUs) against 1 machine (8 A100 GPUs). Our experiments show that ScaleFusion surpasses state-of-the-art techniques, achieving an average speedup of 1.36× (up to 1.58×).

1 INTRODUCTION

High-quality AI-generated videos are becoming increasingly accessible thanks to the recent advancements in video-related foundation models such as spatial-temporal diffusion transformers (ST-DiTs) (Zheng et al., 2024; Singer et al., 2022; Ma et al., 2024; Gupta et al., 2023). For example, OpenAI's Sora (Brooks et al., 2024) was the first to demonstrate the capability of generating 1080p 1-minute-long photo-realistic videos from text prompts. As ST-DiT models advance in capability and quality, generating high-resolution long videos is becoming progressively feasible for industrial production.

However, serving diffusion models on modern GPUs is computationally expensive, especially when generating high-resolution long videos. For example, generating a 1080p 4-second-long video using the OpenSora ST-DiT model (1.1B parameters) (Zheng et al., 2024) on a single A100 GPU requires more than 5 minutes. This inference latency has significantly limited applying generating high-resolution long videos to time-sensitive use cases. While recent research has explored innovative modeling paradigms to reduce inference latency (Song et al., 2022; Lu et al., 2022; Song et al., 2023; Liu et al., 2022), we propose to address this challenge through parallel computing, distributing the ST-DiT's workloads across multiple GPUs and parallelizing executions.

Generating high-resolution long videos essentially imposes long sequence workloads on the spatial and temporal attention layers of ST-DiTs. Existing optimizations for serving ST-DiTs employ sequence parallelism (SP) techniques (Liu et al., 2023; Jacobs et al., 2023) to distribute the workloads along sequence dimensions across multiple GPUs for parallel processing (Fang & Zhao, 2024; Zhao et al., 2024). However, these methods suffer from high communication overhead between GPU machines due to lack of computation-communication overlap. As a result, scaling these methods to further reduce ST-DiT inference latency with more compute resources is challenging.

To analyze the communication-overlap challenges of ST-DiTs, we first examine their computation-communication pattern. As shown in Figure 1, a typical ST-DiT architecture consists of alternating spatial and temporal attention layers. Distributed inference of this model on multiple GPUs involves sharding the input tensor along the temporal dimension for parallelizing spatial attention, followed by

[*]This work is done during internship at Amazon Web Services, Santa Clara, USA. [1]University of Toronto [2]Amazon Web Services. Correspondence to: Gennady Pekhimenko <pekhimenko@cs.toronto.edu>.

Proceedings of the 8^{th} MLSys Conference, Santa Clara, CA, USA, 2025.

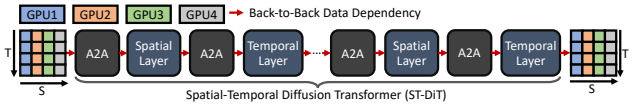

Figure 1. The distributed execution of ST-DiT introduce back-to-back dependencies.

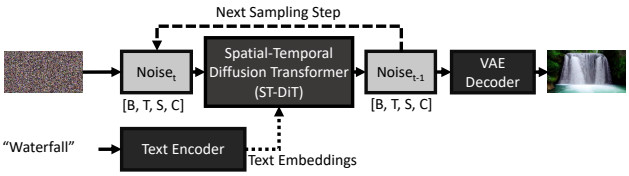

Figure 2. Illustration of video diffusion processes with ST-DiT.

sharding along the spatial dimension for parallelizing temporal attention. All-to-all communication collective is inserted between these two layers to re-shard the tensor (Zhao et al., 2024). In this distributed computation-communication pattern, each operation is serially depdendent on the completion of its predecessor. As measured in our experiments, cross-GPU communication overhead accounts for 30-50% of the ST-DiT's end-to-end inference latency when multiple GPU machines are used, hindering scalability to larger GPU clusters.

To address this challenge, prior research has explored lossy techniques to hide communication overhead in the diffusion process for image generation (Li et al., 2024; Wang et al., 2024). These methods leverage the iterative nature of diffusion models, utilizing activation values from previous iteration to compute activations for the current iteration. While similar approaches could be applied to video generation, they inevitably compromise visual quality due to the reliance on stale information. To our knowledge, no existing method effectively hides communication overhead without sacrificing video generation quality.

In this work, we identify a fundamental property underlying the ST-DiT architecture: spatial-temporal independence. We observe that the spatial attention can be executed independently of the temporal dimension, i.e., treating the temporal dimension as a batch dimension. This property enables the input tensor to the spatial attention to be partitioned along the temporal dimension into multiple slices. As a result, the execution of the spatial attention and subsequent all-to-all operation can be pipelined across these slices, leading to the overlap of computation (spatial attention) and communication (all-to-all). A similar pipeline can be established for the temporal attention and its associated all-to-all operation.

Specifically, we devise intra- and inter-layer communication scheduling algorithms to minimize communication overhead. Intra-layer communication schedules all-to-all operation for each slice to overlap with its associated computation layer (spatial or temporal attention) processing the previous slice. Inter-layer communication further schedules partial all-to-all operation for each slice to overlap with the previous computation layer processing the last slice. By combining these techniques, we significantly reduce the communication latency when scaling the model to multiple GPU machines for the inference of ST-DiTs.

We summarize ScaleFusion's contributions as follows:

- We analyze and quantify the high communication overhead of existing works utilizing SP techniques for accelerating the inference of ST-DiTs on multiple GPU machines, and identify the key property of ST-DiTs: spatial-temporal independence, which can be leveraged to reduce the communication overhead.
- We are the first to address the high communication overhead in the inference of ST-DiTs on multiple GPU machines and develop ScaleFusion consisting of novel intra-layer and inter-layer communication scheduling algorithms to achieve optimized computation-communication overlap.
- We conduct comprehensive benchmarks for ScaleFusion and existing techniques on the inference of Open-Sora's ST-DiT model, and demonstrate that ScaleFusion can achieve speedup of 1.40× on average (up to 1.58×) compared to the state-of-the-art work. In addition, ScaleFusion achieves an average strong scaling of 3.60× on 4 `p4d.24xlarge` machines (32 A100 GPUs) and 1.93× on 2 machines (16 A100 GPUs) against 1 machine (8 A100 GPUs).

2 Background

In this section, we briefly introduce the video diffusion processes, its neural network architectures, and existing works that run them for distributed inference on multiple GPU machines.

Video Diffusion Process with Spatial-Temporal Diffusion Transformers Figure 2 illustrates a typical diffusion process (Rombach et al., 2022; Ho et al., 2022) for video generation with spatial-temporal diffusion transformers (ST-DiT). It iteratively denoises a random noise sample through a sequence of sampling steps, gradually transforming it into a clean video. The number of sampling steps required varies depending on the specific methodology for training or inference, ranging from tens to hundreds. In each sampling step, a neural network predicts the denoising operation. This neural network evaluation, which is the major computational bottleneck, is run for every sampling step. To address this computational challenge, prior research has primarily focused on reducing the number of sampling steps through

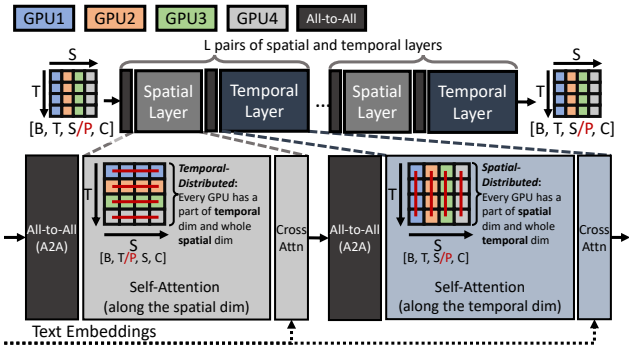

Figure 3. Distributed execution of ST-DiT. For simplicity, the auxiliary layers such as layer normalizations are not shown.

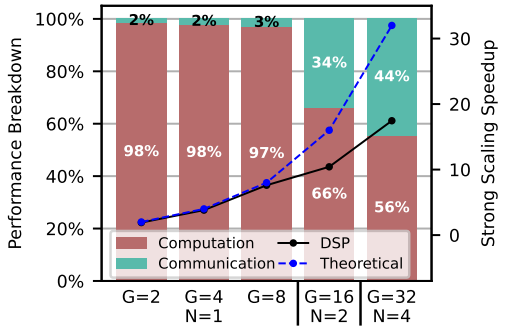

Figure 4. The performance breakdown and strong scaling speedup when executing ST-DiT with the state-of-the-art work (Zhao et al., 2024) on single or multiple GPUs, where N represents the number machines and G represent the total number of GPUs.

algorithmic innovations such as training-free methods (Song et al., 2022; Lu et al., 2022) and novel training paradigms (Song et al., 2023; Liu et al., 2022). In contrast, our work aims to accelerate the individual sampling steps by optimizing the inference system of the neural network through distributed computing.

Among all the neural network architectures for the diffusion process, ST-DiTs have emerged as a promising architecture for high-quality video generation in recent years (Brooks et al., 2024; Ma et al., 2024; Gupta et al., 2023; Yang et al., 2024; Zheng et al., 2024; Singer et al., 2022). The backbone of this architecture was initially proposed by ViViT (Arnab et al., 2021) for video classification tasks. However, the quadratic compute cost scaling of the attention computation in spatial and temporal layers with respect to video resolution and duration poses significant challenges for generating high-resolution long videos. In this work, we optimize the distributed inference for the ST-DiT architecture employing a ViViT backbone with repeated alternating spatial-temporal attention layers as in Figure 1.

Distributed Inference of ST-DiTs on Multiple GPUs Figure 3 demonstrates the distributed inference of a typical ST-DiT containing repeated alternating spatial and temporal layers on multiple GPUs. As the sizes of the spatial and temporal and dimensions scale linearly with the resolution and duration of the videos to be generated, existing systems utilize sequence parallelism (SP) techniques for accelerating the generation of high-resolution long videos (Zheng et al., 2024; Zhao et al., 2024). These systems partition the input tensor along spatial and temporal sequence dimensions, distributing the resulting sharded tensors across multiple GPUs for parallel processing. To compute spatial or temporal attention, which requires global information along the sequence dimension, communication operations are used to gather the sharded tensors, enabling the execution of attention layers. This section reviews the existing state-of-the-art distributed inference technique proposed by (Zhao et al., 2024) for ST-DiTs. The discussion of model parallelism techniques for other models is deferred to Section 6.

Consider an input tensor to an ST-DiT model has a shape of $[B, T, S, C]$, where B denotes batch size, T and S denote the sizes of temporal and spatial sequence dimension, and C denotes the size of hidden dimension. Note that the attention computation of spatial and temporal layers occurs along the dimension of S and T, respectively. To run the model inference with P GPUs in parallel, the input tensor is split along the spatial dimension into P shards, each of which is distributed to a GPU for processing. Note that each shard has a shape of $[B, T, S/P, C]$. We name this layout *spatial-distributed*. As the execution of the spatial attention requires a complete spatial sequence dimension, (Zhao et al., 2024) inserts an all-to-all operation to gather shards from GPUs along the spatial sequence dimension and scatter along the temporal sequence dimension, resulting in a shard of shape of $[B, T/P, S, C]$. We name this layout *temporal-distributed*. Similarly, when executing the following temporal layer, a complete temporal sequence dimension is required. As the input tensor is temporal-distributed after the spatial layer, another all-to-all operation is inserted to transform *temporal-distributed* layout back to the *spatial-distributed* layout, by gathering shards from GPUs along the temporal dimension and scattering along the spatial dimension. After the execution of a pair of a spatial layer and a temporal layer, the output tensor transforms to the original spatial-distributed layout with the same shape as the input tensor, i.e., $[B, T, S/P, C]$. The aforementioned process repeats for the remaining layers.

3 Challenges & Opportunities

While there are works utilizing SP techniques to shard attention layers across multiple GPUs (Fang & Zhao, 2024; Zhao et al., 2024), none of them has addressed high communication overheads when scaling ST-DiT inference across multiple GPU machines to the best of our knowledge. Figure 4

illustrates the computation-communication breakdown of an existing state-of-the-art work (Zhao et al., 2024) generating a 1080p 8 second video. While existing SP techniques has negligible communication overhead (i.e. by only up to 3%) in single-machine settings, they incur huge communication overhead when scaling up to multiple machines. Specifically, these techniques incur 34% time and 44% executing communication operations in 2-machines and 4-machines experiments, leading to 1.53× and 1.83× slowdowns compared to the theoretical strong scaling speedups. This is because GPUs on the same machine are connected with high-bandwidth interconnects such as NVLink or NVSwitch (NVIDIA, b) and their bandwidth is much larger than the cross-machine networks. Thus, even with the state-of-the-art SP techniques, generating long high-resolution ST-DiT does not scale efficiently with multiple GPU machines.

To reduce communication overhead, a common wisdom is to execute the communication operations concurrently with the computation operations and hide the communication latency as much as possible (Aminabadi et al., 2022; Huang et al., 2019; Harlap et al., 2018; Jiang et al., 2024; Jayarajan et al., 2019). However, hiding the communication overhead for ST-DiTs is non-trivial. We elucidate the challenges and opportunities as follows.

Challenge 1: Back-to-back latency between computation and communication operations. As discussed in Section 2, ST-DiT contains alternating spatial and temporal layers and requires all-to-all operations that transform back and forth between the spatial-distributed layout and temporal-distributed layout. As shown in Figure 1, since the input tensor of each all-to-all operation is the output tensor of its previous spatial/temporal layer, and its output tensor becomes the input tensor of the next spatial/temporal layer, executing ST-DiT on multiple GPUs naturally introduce back-to-back dependencies between the communication operations and computation operations. Therefore, it is non-trivial to execute communication and computation operations concurrently to hide the communication overhead.

Opportunity 1: Intra-layer communication-computation overlap. To solve the challenge 1, we observe that the aforementioned back-to-back dependencies only exist along the spatial dimension in spatial layers and along the temporal dimension in temporal layers. Thus, the layer computation operations is totally independent along the other dimension, i.e. temporal dimension in spatial layers and spatial dimension in temporal layers. As a result, we can divide the layer execution into slices where the back-to-back communication-computation dependencies only exist within each slice. By doing so, we create new opportunities for overlapping the communication operations with computation operations between different slices.

Challenge 2: Diminishing speedups with an increased number of slices. Figure 5 shows how the (non-overlapped) communication and computation overhead vary when sharding the layer execution into different number of slices. While increasing the number of slices allows more communication operations to be overlapped and thus reduces the non-overlapped communication overhead (Figure 5a), it also increases the computation overhead (Figure 5b). Since the model is computation bounded as indicated by Figure 4, the increasing computation overhead will cause diminished speedups when increasing the number of slices (Figure 5c). Thus, it is not always beneficial to overlap more communication operations by increasing the number of slices.

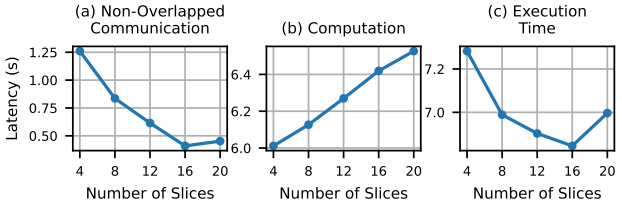

Figure 5. The (a) decreasing non-overlapped communication overhead, (b) the increasing computation overhead, and (c) the overall execution time when increasing the number of slices of layer execution with opportunity 1.

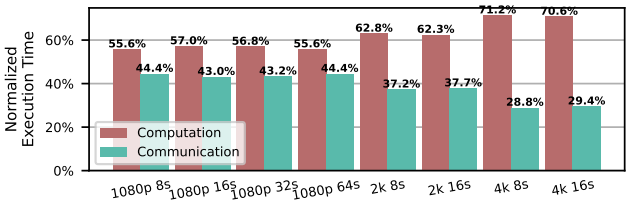

Figure 6. The breakdown of the execution time of spatial layers and temporal layers when generating videos with different resolutions and duration with the state-of-the-art work (Zhao et al., 2024).

Opportunity 2: Inter-layer communication-computation overlap. Figure 6 illustrates the breakdown of the execution time with the state-of-the-art work on the computation and communication operations. We observe that the state-of-the-art implementation of typical video generation workloads are bounded by computation operations. Thus, instead of partitioning both the computation and communication operations by simply increasing the number of slices, we could avoid computation overhead by only further partitioning the communication operations. Even though the overall communication time could be slightly increased, it should be beneficial as long as it can be overlapped with computation operations. Specifically, we consider partition the communication operations and move part of them to overlap with the communication operations in the previous spatial/temporal layer. In this way, we could overlap more communication operations at no cost of computation over-

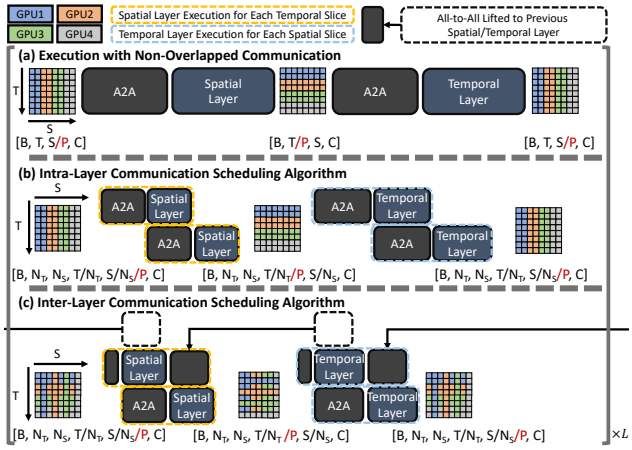

Figure 7. Overview of ScaleFusion's key ideas. For simplicity, the figure demonstrates only the execution of one of the L pairs of spatial and temporal layers in a ST-DiT.

head, thus further reducing the communication overhead.

4 SCALEFUSION

In this work, we propose ScaleFusion, which composes of intra- and inter-layer communication scheduling algorithms.

4.1 Overview

We delinate the overview of the execution of ST-DiTs on multiple GPUs with the state-of-the-art existing work (Zhao et al., 2024) and ScaleFusion in Figure 7. We observe that the existing work suffer from high communication overhead, since they do not overlap the all-to-all communication operations with computations (Figure 7a). Instead, in this paper we propose the following key principle named *spatial-temporal independence*, which allows us to break each temporal and each spatial layer into multiple slices that can be executed independently. Based on this key principle, we propose ScaleFusion that composes of intra- and inter-layer communication scheduling algorithms to achieve efficient computation and communication overlap when executing ST-DiTs on multiple GPUs.

Intra-Layer Communication Scheduling Algorithm As shown in Figure 7b, we schedule communication operations of each slice concurrently with the computation operations of the next slice in both spatial and temporal layers and execute each communication and computation slice in a pipelined fashion. Specifically, when executing the computation of one slice, we schedule the communication of the next slice concurrently by leveraging the asynchronous communication operation. In this way, ScaleFusion is able to overlap communication with computation between difference slices.

Inter-Layer Communication Scheduling Algorithm While intra-layer scheduling algorithm can overlap a large portion of communication with computation operations, the communication of the first slice and the computation on the last slice cannot be overlapped. Note that this issue cannot be simply solved by increasing the number of slices as it incurs computation overhead (see Section 5.3 details). We propose inter-layer communication scheduling algorithm, which further divide the first communication slice into multiple partitions, and schedule communication of a few partitions when executing the computation of the last slice in the previous layer, without blocking the critical execution path.

Through the intra- and inter-layer communication scheduling algorithms, we can achieve almost fully overlapped communication and thus significantly reduce the communication overhead.

4.2 Key Principle: Spatial-Temporal Independence

In the execution of a ST-DiT, we identify that the tokens in temporal dimension are totally independent when executing attention in spatial layers and similarly the tokens in spatial dimension are totally independent when executing attention in temporal layers.

Specifically, assume the input tensor has a shape of $[B, T, S, C]$, where B, T, S, and C represents the size of batch, temporal, spatial, and hidden dimensions respectively. Suppose we have divided the input tensor x along the temporal dimension into three slices, and their corresponding input tensors are denoted as $x_{:,t_1,:}$, $x_{:,t_2,:}$, and $x_{:,t_3,:}$ respectively. Then we can compute the attention in spatial layers independently and then concatenated together, namely,

$$\begin{aligned}&\text{SpatialLayer}(\text{A2A}_{T\to S}(\text{cat}_T(x_{:,t_1,:}, x_{:,t_2,:}, x_{:,t_3,:})))\\ &= \text{cat}_T\begin{pmatrix}\text{A2A}_{T\to S}(\text{SpatialLayer}(x_{:,t_1,:})),\\ \text{A2A}_{T\to S}(\text{SpatialLayer}(x_{:,t_2,:})),\\ \text{A2A}_{T\to S}(\text{SpatialLayer}(x_{:,t_3,:}))\end{pmatrix}\end{aligned} \tag{1}$$

where cat_T denotes tensor concatenate along the temporal dimension and $\text{A2A}_{T\to S}$ denotes the all-to-all operation that scatters the temporal dimension and gathers the spatial dimension.

Similarly, for temporal layers, assume we have divided the input tensors along the spatial dimension into three partitions and denote the corresponding input tensors by $x_{:,:,s_1}$, $x_{:,:,s_2}$, and $x_{:,:,s_3}$, then we have,

$$\begin{aligned}&\text{TemporalLayer}(\text{cat}_S(\text{A2A}_{S\to T}(x_{:,:,s_1}, x_{:,:,s_2}, x_{:,:,s_3})))\\ &= \text{cat}_S\begin{pmatrix}\text{A2A}_{S\to T}(\text{TemporalLayer}(x_{:,:,s_1})),\\ \text{A2A}_{S\to T}(\text{TemporalLayer}(x_{:,:,s_2})),\\ \text{A2A}_{S\to T}(\text{TemporalLayer}(x_{:,:,s_3}))\end{pmatrix}\end{aligned} \tag{2}$$

where cat_S denotes tensor concatenate along the spatial

dimension and $\text{A2A}_{S\to T}$ denotes the all-to-all operation that scatters the spatial dimension and gathers the temporal dimension.

Thus, instead of execute the spatial or temporal layer on the whole input tensors, we could divide the input tensors into multiple slices and execute each slice separately and finally concatenate the results together. This key principle presents potential opportunity for overlapping the communication overhead with computation, for which we propose intra- and inter-layer scheduling algorithms.

4.3 Intra-Layer Communication Scheduling Algorithm

As mentioned in Section 2, when executed with multiple GPUs, each spatial and each temporal layer in ST-DiT incur an all-to-all operation before the execution of the layer itself. To hide the latency of all-to-all operations, we leverage the key principle to divide the layer execution into several slices. Since the execution of each slice is independent, we can execute the all-to-all operation and the layer execution separately of each slice and organize the execution in a way such that the all-to-all operation execution is overlapped with the layer computation from the subsequent slice.

Specifically, as shown in Figure 8a, we ❶ introduce two hyper-parameters N_T ($N_T \leq T$) and N_S ($N_S \leq S$), meaning the number of slices in the temporal and the spatial dimension respectively. We ❷ first rearrange the input tensor x on each GPU to a shape of $[B, N_T, N_S, T/N_T, S/N_S/P, C]$, where B, T, S, and C denote the size of batch, temporal, spatial, and hidden dimension respectively and P denotes the number of GPUs. Note that T and S do not need to be divisible by N_T or N_S. If N_T does not divide T or N_S does not divide S, we just evenly split T and S into N_T and N_S parts respectively, and simply employ a jagged tensor to represent x where each $x_{:,i,j}$ could have different dimensions among all $i \in [0, N_T)$ and $j \in [0, N_S)$. When executing the spatial layers, for communication operations, we ❸ start by only executing the all-to-all operation in the first temporal slice of a spatial layer. Then, immediately after the all-to-all operation in the current temporal slice is finished, we ❹ start executing the all-to-all operation in the following temporal slice. For computation operations, we simply execute the computation operations in the order of the temporal slices. To meet the data dependency, we block the execution of the computation operations until the dependent all-to-all operation has finished. Similarly, for temporal layers, we ❺ execute all all-to-all operations in a back-to-back manner and we block the execution of the computation operations until the dependent all-to-all operation is finished. By dividing the layer execution into different slices, we can overlap part of the all-to-all operations with the computation operations and thus reducing the communication overhead.

Ideally, we only have the all-to-all operation in the first slice and the layer computation operations in the last slice that are not overlapped. Suppose the communication overhead is $Comm_T$ and $Comm_S$ for temporal layers and spatial layers respectively. By using the intra-layer communication scheduling algorithm, if the all-to-all operations in all slices except the last slice are overlapped completely, we could reduce the communication overhead to $Comm_T/N_T + Comm_S/N_S$.

According to the formula above, it is tempting to increase the number of spatial/temporal slices, since it could further reduce the communication overhead. However, in practice we found that using a large number of temporal/spatial slices leads to many small CUDA kernel launches for the computation operations, which causes high computation overhead and in turn slows down the overall execution time (see Section 5.3 for details). As a result, simply applying intra-layer communication scheduling algorithm can only overlap a certain amount of communication operations without incurring computation overhead. To solve this issue, we propose the following inter-layer communication scheduling algorithm to conduct inter-layer communication-computation overlap.

4.4 Inter-Layer Communication Scheduling Algorithm

While intra-layer communication scheduling algorithm can overlap the all-to-all communication operations and the layer computation operations, the all-to-all operation in the first slice and the layer computation in the last slice is still not overlapped. To further reduce the communication overhead without incurring computation overhead, we introduce the inter-layer communication scheduling algorithm, which overlaps part of the all-to-all communication in the first slice with the computation operations in the last slice.

The key insight behind inter-layer communication scheduling algorithm is that we could lift part of the all-to-all operations to overlap with the computation operations in the last slice of the previous spatial or temporal layer. Specifically, in the execution of a temporal layer, we find that each all-to-all operation can be further decomposed along the temporal dimension. For example, assume we have partitioned both the spatial and temporal dimension into three slices, which we denote as s_1, s_2, s_3, and t_1, t_2, t_3 respectively. Then, for *each* spatial partition s_i, we can ❻ decompose the all-to-all operation into three partitions,

$$\begin{aligned} &A2A_{T\to S}(cat_T(x_{:,t_1,s_i}, x_{:,t_2,s_i}, x_{:,t_3,s_i}))) \\ &= cat_T \begin{pmatrix} A2A_{T\to S}(x_{:,t_1,s_i}), \\ A2A_{T\to S}(x_{:,t_2,s_i}), \\ A2A_{T\to S}(x_{:,t_3,s_i}) \end{pmatrix} \end{aligned} \tag{3}$$

By doing so, we can ❼ lift the first two communication

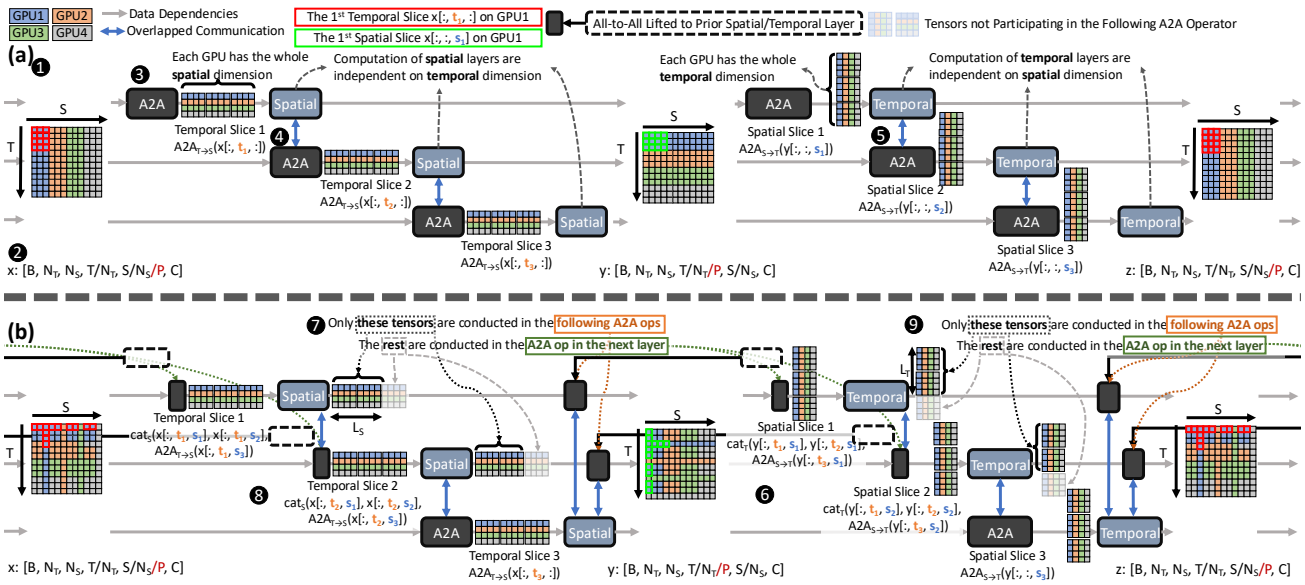

Figure 8. Illustration of (a) the intra-layer communication scheduling algorithm and (b) the inter-layer communication scheduling algorithm within one spatial layer and a (subsequent) temporal layer of a ST-DiT, when configuring $N_T = N_S = 3$.

partitions to overlap with the computation operations in the last slice of the previous spatial layer. More specifically, if we compare the middle tensor between 8a and 8b, only the 3-rd, the 6-th, the 9-th, and the 12-th row are the same. This is because the inter-layer communication scheduling would start conducting the all-to-all operations on the other rows earlier to overlap with the computation operations. As shown in the green boxes in Figure 8b, the tensor $x[:, :, s_1]$ on GPU1 could store data from the other spatial slices. such misplacement will be fixed by the all-to-all operations in the following layer.

Similarly, in the execution of a spatial layer, we could also ❽ divide the all-to-all operation in the first temporal slice into three partitions along the spatial dimension. Namely, for *each* temporal partition t_i, we have,

$$A2A_{S\to T}(cat_S(x_{:,t_i,s_1}, x_{:,t_i,s_2}, x_{:,t_i,s_3})) = cat_S \begin{pmatrix} A2A_{S\to T}(x_{:,t_i,s_1}), \\ A2A_{S\to T}(x_{:,t_i,s_2}), \\ A2A_{S\to T}(x_{:,t_i,s_3}) \end{pmatrix} \quad (4)$$

And we can also ❾ lift two communication partitions to overlap with the computation operations in the last slice of the previous temporal layer. Note that this process is only not applicable at the first (spatial) layer of ST-DiTs, simply because there is no previous temporal layer to overlap with. However, its communication overhead is negligible compared to the end-to-end execution time of the whole model.

In ScaleFusion, we design the inter-layer communication algorithm that generalizes the above example by defining another two hyperparameters, L_T and L_S, representing the numbers of the all-to-all operation partitions in the first slice of each spatial/temporal layer lifted to overlap with the computation operations in the last slice of the previous temporal/spatial layer. The example above is a special case of inter-layer communication scheduling algorithm by choosing $L_T = L_S = 2$. Intuitively, L_T and L_S controls the amount of communication operations in the first slice of the spatial/temporal layer that are lifted to the previous layer. With a larger value of L_T and L_S, we have more communication operations in the spatial/temporal moved to the previous temporal/spatial layer. Besides, by choosing a different L_T and L_S, ScaleFusion is adaptable to different video generation workloads. If the temporal layer has longer computation time than the spatial layer, we choose $L_T \geq L_S$, which creates more opportunities to overlap communication operations with computation operations in the temporal layer. On the other hand, if the spatial layer has longer computation time, we simply choose $L_T < L_S$.

Note that the inter-layer communication scheduling algorithm can not only further overlap the communication operations that are not able to be overlapped by the intra-layer communication scheduling algorithm, it will not increase the computation overhead when applied on top of the intra-layer communication scheduling algorithm since we do not increase the number of partitions of computation operations. Thus, applying the inter-layer communication scheduling algorithm on top of the intra-layer computation scheduling algorithm can solve both challenge 1 and challenge 2 as described in Section 3.

Specifically, if the computation operations in each slice takes longer time than the communication operations, by using inter-layer communication scheduling algorithm, we can reduce the communication overhead to $(Comm_T + Comm_S)/N_S N_T$, since the only communication operations that are not overlapped with computations are the first spatial/temporal communication partition in the first slice of each temporal/spatial layer.

Additionally, by slicing the all-to-all operations, we also reduce the peak memory usage (see Section 5.2 for details). This is because each all-to-all operation requires a temporary buffer to receive the tensor values from other GPUs. Therefore, by dividing the all-to-all operations in different slices, we can release the corresponding temporary buffer after each slice of the all-to-all operation is finished, resulting in lower peak memory usages.

Overall, ScaleFusion maximally overlaps the communication operations with the computation operations in all evaluated video generation workloads, reduce the communication cost and peak memory footprint, and thus enable efficient high resolution long video generation on multiple GPUs.

4.5 Pseudo-Code Implementation

We describe ScaleFusion's pseudo-code implementation in Algorithm 1. To support asynchronous all-to-all operations, we create two CUDA streams for computation operations and communication operations respectively. For simplicity, we abstract out our implementation in a way where that calling an asynchronous all-to-all operation produces a CUDA event by waiting on which we can block the default CUDA stream (i.e. the computation stream) and get the return value of the all-to-all operation. Particularly, in the code each $E^S_{i,j}$ and each $E^T_{i,j}$ denotes a CUDA event that represents the status of the (asynchronous) all-to-all operation scheduled for the spatial or the temporal layer respectively.

Specifically, to keep the algorithm implementation and our description clean, we initiate the first layer's communication in line 4 to line 6 as if we had a previous temporal layer. For each spatial layer of the video diffusion model, we first initiate the communication that has not handled in the last temporal layer (line 8 to line 10). Then, we wait on all the events and concatenate the result tensors along the spatial dimension (line 12) and execute the spatial layer (line 13). We will then start conduct all-to-all operation for the next temporal layer on the output tensor y according to the inter-layer communication scheduling algorithm (line 14 to line 15). The execution of temporal layers would be similar except that if it is the last layer, we do not conduct communication for the next spatial layer (line 22).

Note that our implementation does not explicitly ensure the execution order of all-to-all operations. This is because by default each CUDA stream will execute each kernel in the kernel launch order and we only use one CUDA stream for all all-to-all operations. Thus, as long as we launch the communication operation in order, we do not have to synchronize between different communication operations to ensure the execution order.

Algorithm 1 ScaleFusion's implementation on each GPU.

Inputs: The input tensor x of shape $[B, T, S/P, C]$, the video diffusion model $\mathcal{M}$, the number of temporal and spatial slices, N_T and N_S, and the number of partitions of the first temporal/spatial slice L_T and L_S.
Outputs: The output tensor of shape $[B, T, S/P, C]$.

1: Create an empty tensor y of shape $[B, T/P, S, C]$
2: Rearrange x to shape $[B, N_T, N_S, T/N_T, S/N_S/P, C]$
3: Rearrange y to shape $[B, N_T, N_S, T/N_T/P, S/N_S, C]$
4: **for** $i \in [0, N_S)$ **do**
5: **for** $j \in [0, L_T)$ **do**
6: $E^S_{j,i} \leftarrow \text{ASYNCA2A}_{T\to S}(x_{:,j,i})$
7: **for** SpatialLayer, TemporalLayer **in** $\mathcal{M}$ **do**
8: **for** $i \in [L_T, N_T)$ **do**
9: **for** $j \in [L_S, N_S)$ **do**
10: $E^S_{i,j} \leftarrow \text{ASYNCA2A}_{T\to S}(x_{:,i,j})$
11: **for** $i \in [0, N_T)$ **do**
12: $tmp \leftarrow \text{cat}_S(\text{Wait}(E^S_{i,0}, \ldots, E^S_{i,N_S-1}))$
13: $y_{:,i,:} \leftarrow \text{SPATIALLAYER}(tmp)$
14: **for** $j \in [0, L_S)$ **do**
15: $E^T_{i,j} \leftarrow \text{ASYNCA2A}_{S\to T}(y_{:,i,j})$
16: **for** $i \in [L_S, N_S)$ **do**
17: **for** $j \in [L_T, N_T)$ **do**
18: $E^T_{j,i} \leftarrow \text{ASYNCA2A}_{S\to T}(x_{:,j,i})$
19: **for** $i \in [0, N_S)$ **do**
20: $tmp \leftarrow \text{cat}_T(\text{WAIT}(E^T_{0,i}, \ldots E^T_{N_T-1,i}))$
21: $x_{:,:,i} \leftarrow \text{TEMPORALLAYER}(tmp)$
22: **if** TemporalLayer is the last layer of $\mathcal{M}$ **then**
23: **break**
24: **for** $j \in [0, L_T)$ **do**
25: $E^S_{j,i} \leftarrow \text{ASYNCA2A}_{T\to S}(x_{:,j,i})$
26: Rearrange x to shape $[B, T, S/P, C]$
27: **return** x

5 EVALUATION

5.1 Experiment Setups

Baselines We compare ScaleFusion with two existing implementations for ST-DiT: (1) OpenSora uses DeepSpeed-Ulysses (Jacobs et al., 2023) utilizing all-to-all operation to transpose between the head and the spatial dimensions; (2) DSP (Zhao et al., 2024) proposes a new SP technique that reduces communication volume by transposing between the

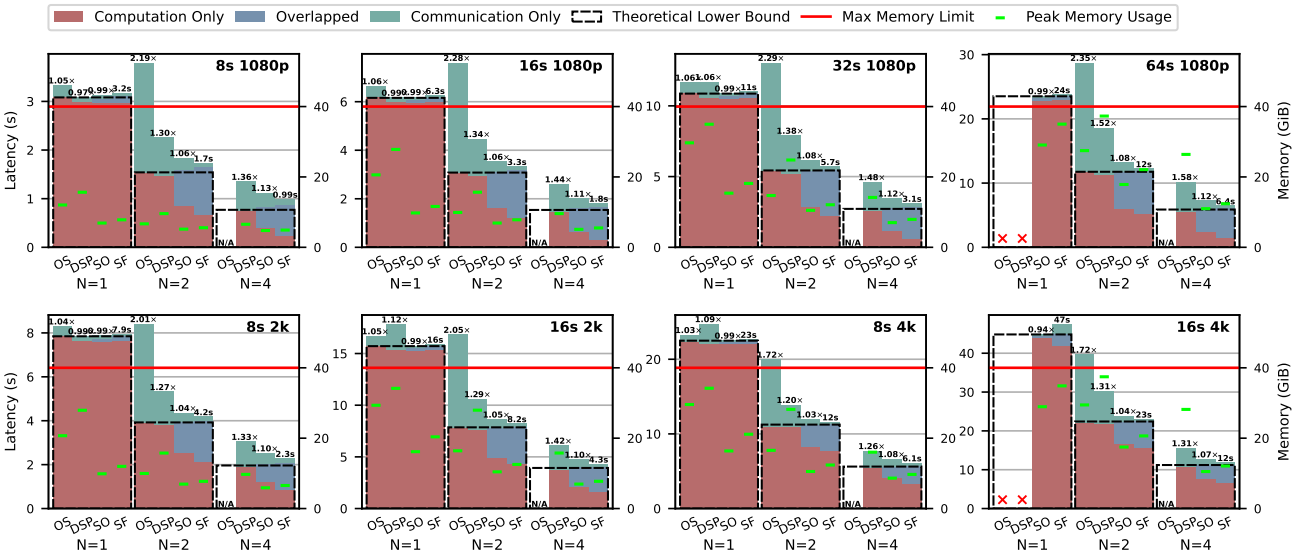

Figure 9. The end-to-end performance of ScaleFusion compared to prior works, where "×" denotes OOM errors, "N/A" means not applicable, and N represents the number of GPU machines. `OS` stands for OpenSora implementation with DeepSpeed-Ulysses (Jacobs et al., 2023), `DSP` stands for the dynamic sequence parallelism (Zhao et al., 2024), `SO` and `SF` stands for ScaleFusion (this work) with only intra-layer communication scheduling algorithm enabled and with both intra- and inter-layer communication scheduling algorithms enabled, respectively.

temporal and spatial dimensions.

Hardware & Software We conduct all experiments on Amazon `p4d.24xlarge` machines that are commonly used in practice for large model inference. Each machine is equipped with 8 NVIDIA A100 GPUs (40 GiB) with NVIDIA Driver 535.183.01, CUDA 12.2, NCCL v2.19.3 and PyTorch v2.2.2 installed. The GPUs within the same machine are connected with NVSwitch (NVIDIA, b) and the machines are connected by AWS Elastic Fabric Adapter (AWS) with 400 Gbps cross-machine network bandwidth.

Neural Networks We adopt the open-source implementation of ST-DiT, OpenSora v1.2 (Zheng et al., 2024) with memory efficient attention enabled (Dao et al., 2022; Dao, 2023), as our codebase for all experiments. Note that for both ScaleFusion and all baselines, we will use classifier-free guidance (Ho & Salimans, 2022) approach to generate the video, which makes the batch size to be 2.

Measurements For runtime latency measurement, we gather the traces produced by NVIDIA Nsight System (NVIDIA, a) when running each experiments. Then calculate the runtime and detailed performance breakdown through analyzing the traces. Note that the end-to-end performance includes all communication and computation operations in the ST-DiT. Particularly, it includes the communication operation of the first partition of the first (spatial) layer when using ScaleFusion. For the theoretical lower bound time (if presented), we simply use the minimum time among all methods in single-machine experiments. In multi-machine experiments, the theoretical lower bound is the single-machine scaling time divided by the number of machines. For peak memory usage measurement, we use `torch.cuda.max_memory_reserved`.

Hyperparameter Choices We use $N_T = N_S = 4$ and $L_T = 1$ and $L_S = 3$ by default since we found these hyperparameters can already hide the communication overhead pretty well in all our experiments with a variety of video generation workloads, as discussed in Section 5.3. However, we also make ScaleFusion configurable by providing interfaces that allow users to set these hyperparameters flexibly.

5.2 End-to-End Performance Evaluation

Figure 9 shows the end-to-end performance of prior works and ScaleFusion. We show normalized speedups of ScaleFusion over OpenSora (`OS`), DSP (`DSP`), and ScaleFusion with only intra-layer communication scheduling algorithm (`SO`), and we show the exact latency numbers for ScaleFusion (`SF`) on their corresponding the bar labels. Note that the OpenSora default implementation (`OS`) cannot scale up to 4 GPU machines. This is because OpenSora uses DeepSpeed-Ulysses (Jacobs et al., 2023) whose parallelism are limited by the number of heads. Since the benchmarked ST-DiT has a small number of heads (i.e., 16), OpenSora's ST-DiT cannot scale up to more than 16 GPUs and thus is not feasible to run on 4 machines of 32 GPUs. We make the following conclusions.

Comparison to Prior Works We found that both Open-Sora and DSP have high communication overhead, which is not hided by the computation. In contrast, ScaleFusion can almost hide all communication costs and achieves a superior speedup by 1.32× on average (up to 1.52×) in 2-machines experiments, and by 1.40× on average (up to 1.58×) in 4-machines experiments over the state-of-the-art work (Zhao et al., 2024). Besides, we also found that as the videos become longer and longer, ScaleFusion can achieve more speedup over prior works, demonstrating its effectiveness for speeding up long high-resolution video generation.

Strong Scaling In our experiments, strong scaling measures the inference speedup as the number of GPUs increases while maintaining a constant workload. We found that prior works suffer from high communication overhead when scaling up to more than one GPU machines. In contrast, ScaleFusion has minimum communication overhead in all scenarios with an average strong scaling of 1.93× in the experiments of 2 machines and 3.60× of 4 machines against 1 machine.

Weak Scaling By comparing the latencies of generating a video of 8s 1080p with 1 GPU machine (3.2s), 16s 1080p with 2 GPU machines (3.3s), and 32s 1080p with 4 GPU machines (3.1s) in Figure 9, we roughly[1] calculate the weak scaling efficiency of ScaleFusion being 97% with 2 GPU machines and 103% with 4 GPU machines respectively.

Speedup Breakdown We observe that using only the intra-layer communication scheduling algorithm can provide 1.26× speedup on average (up to 1.41×), and enabling inter-layer communication scheduling algorithms can provide another 1.08× speedup on average (up to 1.13×) on top of the intra-layer communication scheduling algorithm. Thus, we conclude that both the intra- and inter-layer communication scheduling algorithms are needed to achieve the best performance in ScaleFusion.

Peak Memory Usage We conclude that while both Open-Sora and DSP suffers from out-of-memory (OOM) errors when generating high resolution or long video generation (e.g. when generating 4k videos for 16 seconds), ScaleFusion can generate such videos without hitting OOM errors. ScaleFusion achieves a 1.28× peak memory usage reduction on average (up to 1.43×) compared to OpenSora and 1.87× reduction on average (up to 2.33×) compared to DSP.

Overall, we conclude that ScaleFusion can achieve a significant speedup over all prior works, minimize the communication overhead, and thus enable efficient high-resolution long video generation.

[1]Note that the workload size on each GPU is proportional to T^2S+S^2T where T and S stands for the sequence length of video duration and resolution respectively. Since S is much larger than T in our experiments, doubling T would also roughly doubling the workload size.

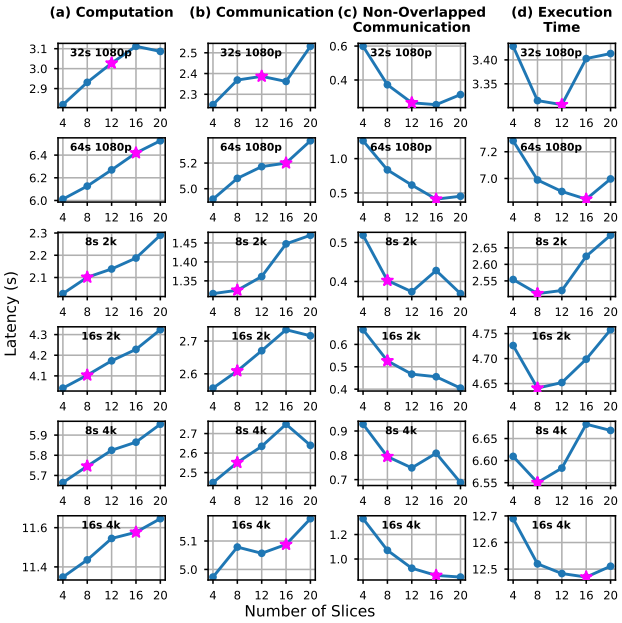

Figure 10. Performance of ScaleFusion with different number of slices when only intra-layer communication scheduling algorithm is enabled. The X-axis ticks represent $N_T = N_S$. The star stands for the optimal configuration.

5.3 Sensitivity Study on ScaleFusion's Hyperparameters

Intra-Layer Communication Scheduling Algorithm Figure 10 shows how the computation time and the non-overlapped communication time varies across different number of slices with only intra-layer scheduling algorithm enabled. We make three conclusions from the figure. First, we conclude that sharding the layer execution into multiple slices incurs both computation and communication overhead (Figure 10a and 10b) that increases with a larger number of slices. Second, we find that non-overlapped communication time is decreasing with larger number of slices (Figure 10c). This is because only the first all-to-all operation in the first slice of each layer is not overlapped which decreases with an increasing number of slices. Third, we notice that while the total time is not always decreasing with a larger number of slices (Figure 10d). Compared to the default hyperparameters we are using, i.e. $N_T = N_S = 4$, the best configuration only brings an additional speedup of 2.7% on average. Thus, we conclude that simply varying its hyperparameters cannot achieve the same level of speedup compared to the case where both intra- and inter-layer communication scheduling algorithms are enabled, i.e. by 1.08× speedup on average.

Inter-Layer Communication Scheduling Algorithm For inter-layer communication scheduling algorithm, we always

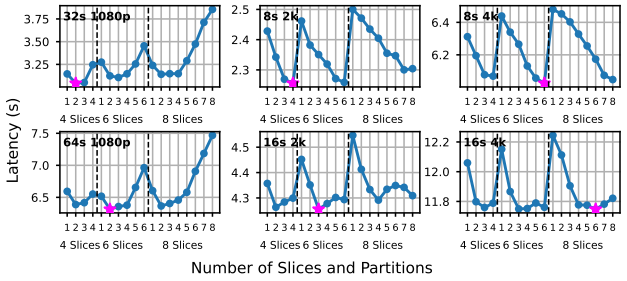

Figure 11. The end-to-end execution time of ScaleFusion with different number of slices when both intra- and inter-layer communication scheduling algorithms are enabled. The first level ticks of X-axis represent L_S and the second level ticks denotes the number of slices, i.e., $N_T = N_S$. The star stands for the optimal configuration.

set $L_T = 1$ and vary L_S since the computation operations in temporal layers are always shorter than the spatial layers in all experiments based on our measurements. Thus, it is always beneficial to lift communication operations from temporal layers to spatial layers but not the other way.

Figure 11 illustrates how the overall execution time varies with different hyperparameters in ScaleFusion. Compared to the default hyperparameters we are using, i.e. $N_T = N_S = 4$, $L_T = 1$, $L_S = 3$, the best configuration only brings 0.7% speedup on average. Thus, we conclude that while the inter-layer communication scheduling algorithm is sensitive to the specific hyperparameters, simply using the ScaleFusion's default hyperparameters can achieve close-to-optimal performance across a variety of scenarios.

6 Related Works

Sequence Parallelism To efficiently serving transformer-based large foundation models on multiple GPUs with long sequence length, different SP techniques have been studied to efficiently distribute the execution of attention layers on multiple GPUs.

DeepSpeed-Ulysses (Jacobs et al., 2023) leverages all-to-all operation to gather and scatter tensors between sequence and attention head dimensions without computation-communication overlap. RingAttention (Liu et al., 2023) divides the query, key, and value tensors along the sequence dimension into multiple partitions and assigns each partition to a single GPU. RingAttention sends and receives each key and value tensor partition from and to its neighboring GPU in a ring-like manner and executes the computation concurrently with the current key and value tensor partition. While RingAttention can overlap the computation with communication, the communication volume is much larger than DeepSpeed Ulysses. This limits its scalability on multiple GPU machines where the inter-machine bandwidth is constrained. DSP (Zhao et al., 2024) optimizes attention operations with multiple sequence dimensions and proposes to apply a similar idea to DeepSpeed-Ulysses. It uses all-to-all operation to gather and scatter tensors between different sequence dimensions. However, unlike ScaleFusion, these two techniques still incur high communication overhead in end-to-end inference.

Pipeline Parallelism Pipeline parallelism (PP) (Huang et al., 2019; Jayarajan et al., 2019; Harlap et al., 2018) partition models into subsets of layers and assign the subsets to multiple GPUs. In execution, PP shards the input tensor along the batch dimension into micro-batches. Each GPU processes one micro-batch and transfers the output to the next GPU in a pipelined manner to keep all GPUs busy. PP works orthogonal with ScaleFusion, since both the inter- and the intra-layer communication scheduling algorithm do not modify the batch dimension and thus can be applied in conjunction with PP.

Lossy Optimizations for Diffusion Models To reduce the communication overhead, prior works have found that diffusion models are resilient to stale inputs, meaning that even if using the outdated input tensors from previous sampling step, diffusion models can still generate images with good quality (Li et al., 2024; Wang et al., 2024). This property breaks the data dependency between consecutive layers, therefore allows concurrent executions of computation and communication to hide communication overhead. However, the accuracy of these methods decrease with an increasing number of GPUs. This is because using more devices implies a larger portion of input tensors are outdated. The quality of the generated contents could further degrade with more GPUs (Table 1 in Li et al. (2024)). Thus, directly applying these methods to ST-DiTs suffers from degraded quality in practice when scaling up to a large number of GPUs. In contrast, ScaleFusion offer lossless algorithm for the distributed inference of ST-DiTs while achieving strong scalability on multiple GPU machines.

7 Conclusion

In this paper, we propose ScaleFusion to optimize communication overhead in ST-DiTs. We identify the spatial-temporal independence that has not been utilized in prior works. We propose both intra-layer and inter-layer communication scheduling algorithms to minimize the communication overhead through computation-communication overlap. Our evaluations show that ScaleFusion can achieve an average speedup of 1.36× (up to 1.58×) speedup compared to prior works. We conclude that ScaleFusion can significantly reduce the communication overhead and scale distributed inference ST-DiTs to multiple GPU machines for high-resolution long video generation.

REFERENCES

Aminabadi, R. Y., Rajbhandari, S., Zhang, M., Awan, A. A., Li, C., Li, D., Zheng, E., Rasley, J., Smith, S., Ruwase, O., and He, Y. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022. URL https://arxiv.org/abs/2207.00032.

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. Vivit: A video vision transformer, 2021. URL https://arxiv.org/abs/2103.15691.

AWS. Elastic Fabric Adapter. URL https://aws.amazon.com/hpc/efa/.

Brooks, T., Peebles, B., Holmes, C., DePue, W., Guo, Y., Jing, L., Schnurr, D., Taylor, J., Luhman, T., Luhman, E., Ng, C., Wang, R., and Ramesh, A. Video generation models as world simulators. 2024. URL https://openai.com/research/video-generation-models-as-world-simulators.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Fang, J. and Zhao, S. Usp: A unified sequence parallelism approach for long context generative ai, 2024. URL https://arxiv.org/abs/2405.07719.

Gupta, A., Yu, L., Sohn, K., Gu, X., Hahn, M., Fei-Fei, L., Essa, I., Jiang, L., and Lezama, J. Photorealistic video generation with diffusion models, 2023. URL https://arxiv.org/abs/2312.06662.

Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., and Gibbons, P. Pipedream: Fast and efficient pipeline parallel dnn training, 2018. URL https://arxiv.org/abs/1806.03377.

Ho, J. and Salimans, T. Classifier-free diffusion guidance, 2022. URL https://arxiv.org/abs/2207.12598.

Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., and Fleet, D. J. Video diffusion models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 8633–8646. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/39235c56aef13fb05a6adc95eb9d8d66-Paper-Conference.pdf.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M. X., Chen, D., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2019. URL https://arxiv.org/abs/1811.06965.

Jacobs, S. A., Tanaka, M., Zhang, C., Zhang, M., Song, S. L., Rajbhandari, S., and He, Y. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models, 2023. URL https://arxiv.org/abs/2309.14509.

Jayarajan, A., Wei, J., Gibson, G., Fedorova, A., and Pekhimenko, G. Priority-based parameter propagation for distributed dnn training. In Talwalkar, A., Smith, V., and Zaharia, M. (eds.), *Proceedings of Machine Learning and Systems*, volume 1, pp. 132–145, 2019. URL https://proceedings.mlsys.org/paper_files/paper/2019/file/3ed923f9f88108cb066c6568d3df2666-Paper.pdf.

Jiang, C., Tian, Y., Jia, Z., Zheng, S., Wu, C., and Wang, Y. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping, 2024. URL https://arxiv.org/abs/2404.19429.

Li, M., Cai, T., Cao, J., Zhang, Q., Cai, H., Bai, J., Jia, Y., Li, K., and Han, S. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7183–7193, June 2024.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context, 2023. URL https://arxiv.org/abs/2310.01889.

Liu, X., Gong, C., and Liu, Q. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022. URL https://arxiv.org/abs/2209.03003.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps, 2022. URL https://arxiv.org/abs/2206.00927.

Ma, X., Wang, Y., Jia, G., Chen, X., Liu, Z., Li, Y.-F., Chen, C., and Qiao, Y. Latte: Latent diffusion transformer for video generation, 2024. URL https://arxiv.org/abs/2401.03048.

NVIDIA. NVIDIA NSight Systems, a. URL https://developer.nvidia.com/nsight-systems.

NVIDIA. NVLink and NVLink Switch, b. URL https://www.nvidia.com/en-us/data-center/nvlink/.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2022. URL https://arxiv.org/abs/2112.10752.

Singer, U., Polyak, A., Hayes, T., Yin, X., An, J., Zhang, S., Hu, Q., Yang, H., Ashual, O., Gafni, O., Parikh, D., Gupta, S., and Taigman, Y. Make-a-video: Text-to-video generation without text-video data, 2022. URL https://arxiv.org/abs/2209.14792.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models, 2022. URL https://arxiv.org/abs/2010.02502.

Song, Y., Dhariwal, P., Chen, M., and Sutskever, I. Consistency models, 2023. URL https://arxiv.org/abs/2303.01469.

Wang, J., Fang, J., Li, A., and Yang, P. Pipefusion: Displaced patch pipeline parallelism for inference of diffusion transformer models, 2024. URL https://arxiv.org/abs/2405.14430.

Yang, Z., Teng, J., Zheng, W., Ding, M., Huang, S., Xu, J., Yang, Y., Hong, W., Zhang, X., Feng, G., et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.

Zhao, X., Cheng, S., Chen, C., Zheng, Z., Liu, Z., Yang, Z., and You, Y. Dsp: Dynamic sequence parallelism for multi-dimensional transformers, 2024. URL https://arxiv.org/abs/2403.10266.

Zheng, Z., Peng, X., Yang, T., Shen, C., Li, S., Liu, H., Zhou, Y., Li, T., and You, Y. Open-sora: Democratizing efficient video production for all, March 2024. URL https://github.com/hpcaitech/Open-Sora.